Ideen für die Prüfung

Mittwoch, 14. Dezember 2022 22:03

Aus dem Datenblatt des Displays ST7735R ist folgendes zu finden:

ST7735R

9.8 Data Color Coding

9.8.1 8-bit Parallel Interface (IM2, IM1, IM0= "100")

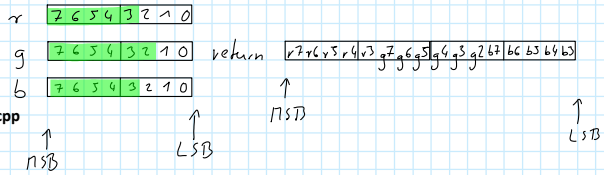
Different display data formats are available for three Colors depth supported by listed below.
- 4k colors, RGB 4,4,4-bit input.
- 65k colors, RGB 5,6,5-bit input.
- 262k colors, RGB 6,6,6-bit input.

Was macht folgende Funktion? Quelle: Adafruit_GFX.cpp Zeile 507

1) **Erstellen Sie ein Diagramm in dem erkenntlich wird, welche Bits der Eingabedaten wie in der Ausgabe eingehen.**
2) **Welche Bits der Eingabedaten werden nicht benutzt**
3) **In welchen Zusammenhang steht das Datenblatt mit dieser Funktion?**
4) **Welchen return-Wert liefert die Funktion bei r = 0x0100, g = 0x1141, b =0x0020**
5) **Erstellen Sie eine Funktion** uint16_t newColor666(uint8_t r, uint8_t g, uint8_t b) **die einen Farbwert im dritten Farbformat des ST7735R Displays zurückliefert.**

Ausgabe Eingabe

```
uint16_t Adafruit_GFX::newColor(uint8_t r, uint8_t g, uint8_t b)
{
 return ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | (b >> 3);
}
```



Was macht diese Funktion? Quelle: Adafruit_ST7735.cpp

```
inline uint16_t swapcolor(uint16_t x) {
 return (x << 11) | (x & 0x07E0) | (x >> 11);
}
```

In dem Wärmebildkamera Projekt ist in der innersten Schleife folgender Code zu finden:

```
HSVtoRGB (R, G, B, hue, 1, .5);
TFTscreen.stroke (RGB(R * 255, G * 255, B * 255));  //-->Adafruit_GFX::newColor Adafruit_GFX.cpp Zeile 507
TFTscreen.point (xi*ZOOM+xd + 1, yi*ZOOM+yd + 75 + 1); //-->void Adafruit_ST7735::drawPixel(int16_t x, int16_t y, uint16_t color)  Adafruit_GFX.cpp Zeile 463
```

Die Funktion TFTscreen.stroke **ruft letztlich** Adafruit_GFX::newColor **auf. Verwendet wird die Farbe aber erst im letzten Funktionsaufruf** TFTscreen.point **die wiederum** Adafruit_ST7735::drawPixel **aufruft.**

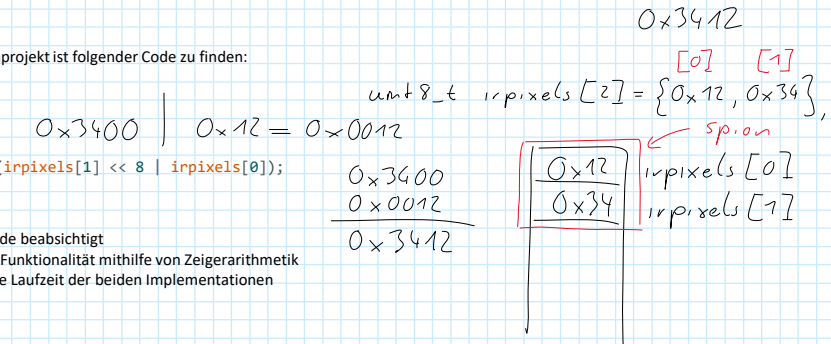
1) **Erläutern Sie welches Verbesserungspotential Sie bei der Handhabung der Farbe erkennen und wie Sie die Implementation abändem würden.**
2) **Messen Sie die Laufzeit der bestehenden Implementierung und speichern Sie die Ergebnisse**
3) **(***Das ist eine echte Herausforderung***) Verbessern Sie den Umgang mit den Farben.**
4) **Messen Sie die Laufzeit den verbesserten Codes und vergleichen Sie die Ergebnisse.**

Aufgabe zur Zeitmessung
Aufgabe zum Zweierkomplement
Aufgabe zu Arrays (Nutzung von Tabellen)

In unserem Wärmebildkameraprojekt ist folgender Code zu finden:

```
uint8_t irpixels[128];

int16_t vir = (int16_t)(irpixels[1] << 8 | irpixels[0]);
```



1) Was wird mit diesem Code beabsichtigt
2) Implementiere dieselbe Funktionalität mithilfe von Zeigerarithmetik
3) Messe und vergleiche die Laufzeit der beiden Implementationen

Oder

1) Bauen Sie Ausgaben ein, so dass die übermittelten Zeichen übersichtlich und hexadezimal auf der seriellen Console ausgegeben werden.
2) Messen Sie nun die Laufzeit der Zeile `int16_t vir = (int16_t)(irpixels[1] << 8 | irpixels[0]);`
3) Speichern Sie dieses Programm unter dem Namen uebung_alt zur Abgabe ab und notieren Sie hier die Anzahl der Taktzyklen, die d iese Codezeile benötigt.
4) In folgenden erstellen Sie das Programm uebung_neu. Bitte achten Sie darauf, dass Sie nicht versehentlich das bereits fertige Programm uebung_alt überschreiben.
   Erstellen Sie ein Programm mit identischer Funktionalität, dass dieselbe Aufgabe möglichst schnell mithilfe von int16_t Zeige r bewältigt. Messen Sie die Laufzeit der neuen Implementation, so dass sie einen Vergleich mit beiden möglichen Implementierungen vornehmen können. W elche Version ist schneller? Um die Ursachen für die Geschwindigkeitsunterschiede eindeutig festzustellen, müsste man den generierten Assem bler Code betrachten. Dies ist leider unter der Arduino Entwicklungsumgebung unnötig kompliziert. Trotzdem versuchen Sie mögliche Ursac hen für die Geschwindigkeitsunterschiede zu erörten.
   -> alte Verison: mindestens zwei Bit-Operationen (<<, |) => mind. zwei Takte
   -> neue Version: Zeiger-cast wird wegoptimiert

   git clone https://github.com/BerndDonner/Microcontrollertechnik.git

```
void setup() {
 // put your setup code here, to run once:
 Serial.begin(9600);

 uint8_t irpixels[2] = {0x12, 0x34};
 int16_t* spion = (int16_t*) &irpixels[0];
 int16_t vir = (int16_t)(irpixels[1] << 8 | irpixels[0]);


 Serial.println("Wert der Variablen");
 Serial.print("Variante Bitoperationen: ");
 Serial.println((int16_t) vir, HEX);
 Serial.print("Variante Zeiger:      ");
 Serial.println(*spion, HEX);
}


void loop() {
 // put your main code here, to run repeatedly:
}
```

Versucht man die HSVtoRGB Funktion unseres Wärmebildkameraprojekts zu optimieren kann man folgende Codezeilen kommen:

```
uint8_t f;
uint8_t q = (0x00ff * (0x0100 - f )) >> 9;
```

1) Wieso liefert diese Codezeile für f = 129 nicht den Wert 63?!
2) Korrigieren sie die Codezeile so, dass hier der gewünschte Wert von 63 ausgegeben wird.

```
    float ai = (acommon + deltaai * pow (2, aiscale)) / pow (2, ( (configreg >> 4) & 0x03));      // Bit 5:4 Config Register
    bi = bi / (pow (2, biscale) * pow (2, 3 - ( (configreg >> 4) & 0x03)));
```

Schreiben Sie die obigen Zeilen als shift-Operationen um

```
/**************************************************************************************************
    @brief  Convert 24-bit RGB to 16-bit 565 RGB
            Adapted from: https://gist.github.com/wirepair/b831cf168882c7013b68c1703bda5250

    @param      rgb24           24-bit RGB

    @return     16-bit 565 RGB
**************************************************************************************************/
uint16_t rgbConvert24to16(uint32_t rgb24){
    uint16_t r = (rgb24 & 0xFF0000) >> 16;
    uint16_t g = (rgb24 & 0xFF00) >> 8;
    uint16_t b = rgb24 & 0xFF;
    uint16_t RGB565 = 0;

    r = (r * 249 + 1014) >> 11;
    g = (g * 253 + 505) >> 10;
    b = (b * 249 + 1014) >> 11;

    RGB565 |= (r << 11);
    RGB565 |= (g << 5);
    RGB565 |= b;

    return RGB565;
}
```

Was müssen wir wissen:

0.) Arduino: setup, loop, Serial.begin(), Serial.print (Debugging), Serial.println, Bediehnung Arduino-oberfläche, Leistungsdaten des ATmega328p
1.) Skopes: {}, lokale, globale Variablen (Micheal Kipp, Howto C-Programmierung)
2.) Funktionen: Parameter, call-by reference, call-by value, return (Micheal Kipp, Howto C-Programmierung, Arrays)
3.) Ganzzahldatentypen: int8_t, int16_t, int32_t, int64_t und unsigned Varianten, Zweierkomplement, Type-cast
4.) Bitoperationen und deren Bedeutung: ~, &, |, ^, >>, <<
5.) Möglichkeiten der Zeitmessung: eigene Routine, millis, micros, wann nimmt man was?, Compiler Optimierungen, volatile
6.) Arrays und Zeiger: Wie liegen Daten im Speicher, Adressoperator &, Zeigeroperator *, call-by reference, spion mit Zeiger-Type-cast

Aufgabe zu Zweierkomplement und überlauf
Aufgabe zu Bitoperationen
Aufgabe zu Arrays (steht im groben)
Aufgabe Zeiger und Zeitmessung (fertig)

Geben Sie an, in welcher Datenstruktur Sie den kompletten Inhalt des TFT Displays (160 horizontal und 128 vertikal) bei horizontaler Ausrichtung speichern könnten. Sie dürfen davon ausgehen, dass die linke obere Ecke die Indexposition (0, 0) trägt. Diese Datenstruktur bezeichnete man übrigens als "framebuffer".

A: `uint16_t frame [128][160]; //Pixel folgen zeilenweise aufeinander 5,6,5 Farbcodierung`

Welches Problem ergibt sich bei der Erstellung dieses Framebuffers auf einem ATmega328p?

A: Bytes: 128*160*2 = 40.960 Bytes, zu wenig Speicher am ATmega328p (nur 2048 Bytes SRAM)

Hier `uint16_t framebuffer[4][6];`
In einem neuen Modus, hat nun die linke untere Ecke die Indexposition (0, 0). Erstellen Sie ein kleines Programm, dass den alten framebuffer auf den neuen so umkopiert, dass der Inhalt für den Benutzer weiterhin korrekt dargestellt wird.

A:

```
//Array Dimensionen werdenvon aussen nach innen angegeben
char text [5][4] = {
  {'a', 'b', 'c', 'd'}, // 0->4,  1->3,  2->2,  3->2,  4->0,  i->4-i
  {'A', 'B', 'C', 'D'},
  {'1', '2', '3', '4'},
  {'w', 'x', 'y', 'z'},
  {'W', 'X', 'Y', 'Z'},
};
char text2 [5][4];
void setup() {
  Serial.begin(9600);
  Serial.println("vorher:");
  for (uint8_t i = 0; i < 5; i++)
  {
    for (uint8_t j = 0; j < 4; j++)
    {
      Serial.print(text[i][j]);
      Serial.print(" ");
    }
    Serial.println("");
  }
  Serial.println("");

  //Umtausch
  for (uint8_t i = 0; i < 5; i++)
  {
    for (uint8_t j = 0; j < 4; j++)
    {
      text2[4 - i][j] = text[i][j];
    }
  }
  //Ausgabe
  Serial.println("nachher:");
  for (uint8_t i = 0; i < 5; i++)
  {
    for (uint8_t j = 0; j < 4; j++)
    {
      Serial.print(text2[i][j]);
      Serial.print(" ");
    }
    Serial.println("");
  }
}

void loop() {
}
```

SPI-Bus Funktionen unberührt lassen

Datentyp aussuchen int8_t bzw. uint8_t
int8_t -128…127
uint8_t 0…255

uint8_t	Binär	int8_t
201	1100 1001	Zweierkomplement rückwärts -1 -> 1100 1000 umdrehen -> 0011 0111 -> -55

```
Serial.begin(9600);
uint8_t a=201;
Serial.println(a); //201
int8_t b=201;
Serial.println(b); //Zweierkomplement also -55
Serial.println((int8_t)a); //andere Möglichkeit zum Bilden vom Zweierkomplement

Zweierkomplement: (~a+1)
```

1. Bug in Programm finden und erklären -> Schleife, die unendlich weiter läuft
2. Arrays und Zeiger int8zuint16altTiming
3. Array erläutern, wie es angeordnet ist, rumschubsen, ausgeben